RoboMonkey: Scaling Test-Time Sampling and Verification for Vision-Language-Action Models

Jacky Kwok[1] **Christopher Agia**[1,†] **Rohan Sinha**[1,†] **Matt Foutter**[1,†]
Shulu Li[2] **Ion Stoica**[2] **Azalia Mirhoseini**[1] **Marco Pavone**[1,3]

[1]Stanford University [2]UC Berkeley [3]NVIDIA Research

https://robomonkey-vla.github.io

Abstract: Vision-Language-Action (VLA) models have demonstrated remarkable capabilities in visuomotor control, yet ensuring their robustness in unstructured real-world environments remains a persistent challenge. In this paper, we investigate test-time scaling through the lens of sampling and verification as means to enhance the robustness and generalization of VLAs. We first demonstrate that the relationship between action error and the number of generated samples follows an exponentiated power law across a range of VLAs, indicating the existence of inference-time scaling laws. Building on these insights, we introduce RoboMonkey, a test-time scaling framework for VLAs. At deployment, RoboMonkey samples a small set of actions from a VLA, applies Gaussian perturbation and majority voting to construct an action proposal distribution, and then uses a Vision Language Model (VLM)-based verifier to select the optimal action. We propose a synthetic data generation pipeline for training such VLM-based action verifiers, and demonstrate that scaling the synthetic dataset consistently improves verification and downstream accuracy. Through extensive simulated and hardware experiments, we show that pairing existing VLAs with RoboMonkey yields significant performance gains, achieving a 25% absolute improvement on out-of-distribution tasks and 9% on in-distribution tasks. Additionally, when adapting to new robot setups, we show that fine-tuning both VLAs and action verifiers yields a 7% performance increase compared to fine-tuning VLAs alone.

1 Introduction

Foundation models, pre-trained on extensive internet-scale data, have demonstrated significant potential in robotics domains. Recent advancements in Vision-Language-Action (VLA) models [1, 2, 3, 4, 5] have shown that scaling up training compute on large-scale robotics datasets [6, 7] can improve their capabilities and generalization. Despite these advancements, VLAs exhibit diverse failure modes during deployment [8, 9], such as imprecise grasping, task progression failure, and collision with surrounding objects. Addressing these limitations could accelerate the deployment of robots in unstructured real-world environments.

Efforts to improve the robustness and generalization of VLAs have gradually shifted from the pre-training to the post-training phase. In the pre-training stage, previous work emphasizes scaling up data collection [10, 7, 11], optimizing training data mixtures [12, 2], and developing model architectures [13, 1, 14, 15] that can be effectively adapted for robot control. More recently, we have observed a paradigm shift toward developments in the post-training phase, e.g., fine-tuning VLAs for multi-step reasoning with chain-of-thought [16, 17, 18] and aligning VLAs with preferences [19, 20, 21]. However, beyond pre-training and post-training, less attention has been paid to scaling the amount of compute used during deployment, as VLA models are typically designed to generate a single action chunk per observation.

Humans naturally allocate more time when encountering challenging problems. For Large Language Models (LLMs), this principle has been validated by applying additional compute at test time [22, 23, 24, 25, 26, 27]. Specifically, repeatedly sampling candidate solutions from a model has been shown to enhance the capabilities of LLMs across multiple domains, including mathematics, coding, chat, and summarization [22, 28, 29]. This raises the question of whether test-time scaling with repeated sampling may also benefit

[†]Equal contribution. Correspondence to: jackykwok@stanford.edu

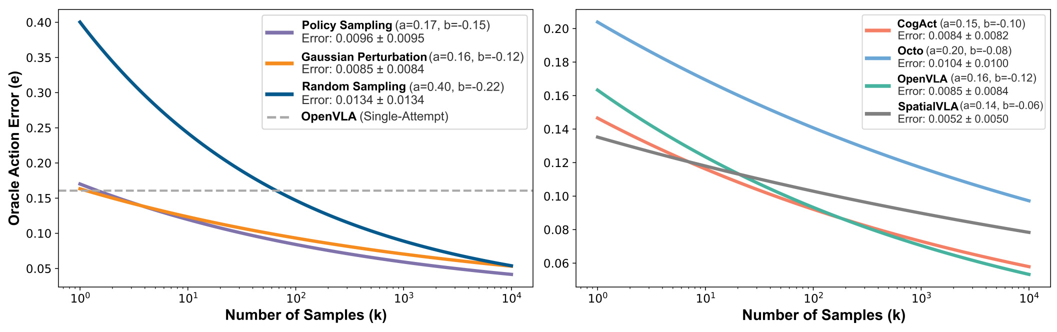

Figure 1: **Inference-Time Scaling Law:** We observe that action error consistently decreases as we scale the number of generated actions across multiple sampling approaches, assuming the presence of an oracle verifier. Repeatedly sampling actions from robot policies, applying Gaussian perturbation to a few sampled actions, and even random sampling all outperform single-attempt OpenVLA. We also find that the relationship between action error and the number of samples generated through Gaussian perturbation follows an approximate power law across a range of VLA models, including CogACT, Octo, OpenVLA, and SpatialVLA. For power law fitting, we model the logarithm of action error e as a function of the number of samples k: $\log(e) \approx \log(a) + b \cdot \log(k)$, where a and b are fitted model parameters.

robotics. More precisely, we ask in this work: given an observation and task instruction, can we improve the precision and robustness of VLAs by repeatedly sampling and verifying actions at deployment?

We answer this question in two parts. First, we systematically investigate the benefits of scaling test-time compute in the domain of static manipulation tasks, using off-the-shelf generalist VLA models as base policies. Through our experiments, we find that the relationship between action error and the number of generated samples follows an exponentiated power law across a range of VLAs, demonstrating the existence of inference-time scaling laws. This finding aligns with the power-law scaling [30, 22] observed in LLMs and suggests that, when paired with a robust verifier, repeated sampling can significantly boost the performance of any off-the-shelf VLA model. Interestingly, different sampling techniques—repeatedly sampling actions from VLAs, Gaussian perturbation applied to a few actions, and random sampling—exhibit a similar scaling pattern. Among these, we find Gaussian perturbation to be the most cost-effective approach and it is therefore adopted in deployment. To our knowledge, our work is the first to characterize inference-time scaling laws for VLAs.

Second, we investigate whether capitalizing on these scaling laws with a learned action verifier can improve policy robustness, guided by the intuition from classic complexity theory that verifying proposals is often easier than generating a solution to a task. To do so, we present a preference-based learning recipe to automatically curate synthetic action comparisons for large-scale imitation learning datasets and use it to train a 7B VLM-based action verifier. Our results show that increasing the synthetic preference dataset size leads to consistent performance improvements. We then introduce our test-time scaling framework, RoboMonkey. During deployment, RoboMonkey samples a small batch of actions from a VLA, applies Gaussian perturbation and majority voting to construct an action proposal distribution, and then uses the fine-tuned VLM-based verifier to select the optimal action. Through extensive evaluations, we demonstrate that pairing existing VLAs with RoboMonkey substantially enhances their precision and robustness.

The contributions of this paper are summarized as follows:

1. We propose efficient methods for action sampling, and demonstrate that the relationship between action error and the number of samples follows an approximate power law across a range of VLAs.
2. We present a scalable pipeline for automatically generating synthetic action preferences along with a method for training a VLM-based action verifier.
3. We show that our test-time scaling framework significantly enhances VLA performance, achieving a 25% absolute improvement in real-world out-of-distribution tasks and 9% on in-distribution SIMPLER environments.
4. We demonstrate that fine-tuning both VLAs and action verifiers yields a 7% performance increase compared to fine-tuning VLAs alone on the LIBERO-Long benchmark.

2 Preliminaries

We consider a Markov Decision Process $M = (\mathcal{S},\ \mathcal{A},\ P,\ R)$, where $\mathcal{S} \subseteq \mathbb{R}^n$ and $\mathcal{A} \subseteq \mathbb{R}^m$ denote the robot's state and action spaces, respectively. In this work, both the state and action spaces are 7-dimensional vector spaces corresponding to the robot's end effector pose and characterized by three translational states $(x,\ y,\ z) \in \mathbb{R}^3$ and three rotational states $(u,\ v,\ w) \in \mathbb{R}^3$, while the last dimension corresponds to a binary state $g \in \{0,\ 1\}$ indicating whether the end effector gripper is open. $a_t = [\Delta x_t,\ \Delta y_t,\ \Delta z_t,\ \Delta u_t,\ \Delta v_t,\ \Delta w_t,\ g_t]'$ indicates the desired magnitude and direction to augment each state variable at time step t. Further, $P(s' \mid s,\ a) \in [0,\ 1]$ represents the robot's non-deterministic transition dynamics from the current state $s \in \mathcal{S}$ with action $a \in \mathcal{A}$ to the candidate state $s' \in \mathcal{S}$, and $R\colon\ \mathcal{S} \times \mathcal{A} \times \mathcal{I} \rightarrow \mathbb{R}$ provides the reward for choosing action $a \in \mathcal{A}$ at state $s \in \mathcal{S}$ under the language instruction $I \in \mathcal{I}$, where $\mathcal{I}$ is the set of possible instructions. Our framework assumes access to a language-conditioned robot policy $\pi_\theta : \mathcal{S} \times \mathcal{I} \rightarrow \mathcal{A}$, parameterized by $\theta \in \mathbb{R}^{|\theta|}$, from which we can sample multiple actions given the state at timestep t and the language instruction $I \in \mathcal{I}$. Additionally, we assume access to a dataset of N_D expert demonstrations performing a suite of manipulation tasks: $\mathcal{D} = \{(\tau^i,\ I^i)\}_{i=1}^{N_D}$, where each demonstration constitutes a valid trajectory $\tau^i = (s_0^i, a_0^i, \ldots, s_T^i)$ under the transition dynamics to time horizon $T \in \mathbb{N}_+$. We further curate an auxiliary dataset $\mathcal{D}_{\text{buf}} \subset \mathcal{D}$ comprising of tuples $(s_t,\ a_t^*,\ I)$, where $a_t^* \in \mathcal{A}$ is the ground-truth action taken by the expert at state s_t under instruction I. We assume that the generalist policy π_θ is fine-tuned to imitate expert demonstrations on this dataset by minimizing the standard imitation learning objective as $\mathcal{L}(\theta; \mathcal{D}) = -\mathbb{E}_{(s_t^j,\ a_t^j,\ I^j) \sim \mathcal{D}}\Big[\log \pi_\theta(a_t^j \mid s_t^j,\ I^j)\Big]$.

3 Inference-Time Scaling Law

The relationship between a VLA's action error and its training compute [31, 32] has been well-documented. However, the potential benefits of scaling test-time compute for VLAs remain largely underexplored. To bridge this gap, we conduct a detailed analysis on the Bridge V2 Dataset [11], examining the relationship between the number of generated samples and action error.

Concretely, we uniformly sample 1,000 $(s,\ a^*,\ I)$ tuples from our auxiliary dataset $\mathcal{D}_{\text{buf}}$. For each tuple, we generate 10,000 actions using various sampling approaches and compute the Normalized Root Mean Squared Error (RMSE) between the ground-truth action a^* and each sampled actions $\{a_1,\ a_2,\ \ldots,\ a_{10{,}000}\}$.

We evaluate three sampling approaches: **Random sampling**: candidate actions are generated by uniformly sampling discrete action tokens for each dimension based on the scheme introduced by Brohan et al. [5] **Policy sampling**: actions are repeatedly sampled from a robot policy $\pi_\theta(a \mid s,\ I)$ with a positive temperature. **Gaussian perturbation**: sampling only 4 actions from a robot policy $\pi_\theta(a \mid s,\ I)$, then fitting a Gaussian distribution from which all candidate actions are drawn (see Section 4.4 for details).

The result is shown in the left plot of Figure 1. Assuming the presence of an oracle verifier that always selects the action with the lowest RMSE, we observed that as we scale the number of generated samples, the action error consistently decreases across all sampling methods. Our key findings are: (1) sampling more than 100 actions uniformly at random outperforms greedy decoding using OpenVLA; (2) using policy sampling to repeatedly generate actions from a VLA consistently yields the lowest action error; and (3) Gaussian perturbation achieves nearly identical performance compared to policy sampling while being computationally more efficient. A comprehensive latency analysis is provided in Section 5.4.

The right plot of Figure 1 demonstrates that scaling the number of generated samples with Gaussian perturbation is effective across various generalist robot policies, including CogACT, Octo, OpenVLA, and SpatialVLA [33, 34, 2, 35]. We find that the relationship between action error and the number of samples often follows an exponentiated power law. Specifically, for OpenVLA, the RMSE decreases by 59.3% when sampling 10,000 actions. Overall, we offer a new perspective on how we might approach general robot foundation models. Rather than framing robot control purely as a generation problem, our results suggest that viewing it through the lens of verification—generating diverse candidates and verifying them—can substantially improve performance. We hope our findings will motivate and guide the development of scalable action verifiers for robot policies.

4 Proposed Approach: RoboMonkey

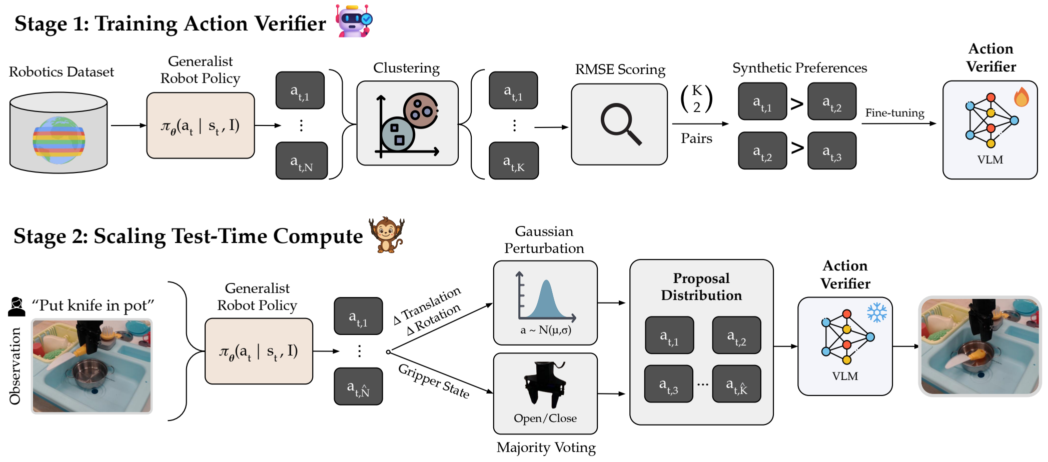

Figure 2: **Stage 1: Training the Action Verifier.** Given an imitation learning dataset, we sample N candidate actions per state from a generalist robot policy, and apply clustering to reduce them to K representative actions. We construct $\binom{K}{2}$ synthetic action comparisons and assign preferences based on the RMSE between each sampled action and the ground-truth action. This synthetic preference dataset is then used to fine-tune a VLM-based action verifier. **Stage 2: Scaling Test-Time Compute.** At deployment, we sample $\hat{N}$ initial actions from the generalist robot policy based on the given task instruction and observation. We fit a Gaussian distribution $\mathcal{N}(\mu, \sigma)$ to the translation and rotation components $(\Delta x, \Delta y, \Delta z, \Delta u, \Delta v, \Delta w)$ of these actions, as introduced in Section 2, and use majority voting to determine the gripper state. This creates an action proposal distribution from which we can efficiently sample candidate actions with negligible overhead. Finally, we use the fine-tuned VLM-based verifier to evaluate these $\hat{K}$ candidate actions and select the optimal action.

4.1 Motivation

After establishing the potential for scaling test-time compute for robotics in Section 3, we now present RoboMonkey, a framework that leverages a learned action verifier to scale test-time compute. We first describe our method for curating a synthetic action preference dataset, followed by reward modeling and inference-time techniques used within RoboMonkey's generate-then-verify pipeline.

4.2 Synthetic Data Generation Pipeline

In this section, we outline our approach for generating synthetic action comparisons, which leverages an existing demonstration dataset $\mathcal{D}$ to produce action pairs with high-quality preference labels without the need for human annotation. Specifically, for each tuple (s_t, a_t^*, I) from our auxiliary dataset $\mathcal{D}_{\text{buf}}$, we use a reference robot policy to generate N candidate actions. To ensure diversity among the samples, we apply clustering algorithms, reducing these candidates to K representative actions. Subsequently, we construct $\binom{K}{2}$ pairwise comparisons and compute the RMSE between each sampled action, $\{a_t^1, a_t^2, ..., a_t^K\}$, and the ground-truth action, a_t^*. Then, the "winning" action, a_t^W, and the "losing" action, a_t^L, between any two actions a_t^i and a_t^j are determined as follows:

$$(a_t^W, a_t^L) = \begin{cases} (a_t^i, a_t^j), & \text{if } \text{RMSE}(a_t^i, a_t^*) < \text{RMSE}(a_t^j, a_t^*), \\ (a_t^j, a_t^i), & \text{otherwise.} \end{cases}$$

We use this procedure to instantiate our action preference dataset $\mathcal{D}_{\text{comp}}$ consisting of tuples $(a_t^W, a_t^L, a_t^*, s_t, I)$. Following Ouyang et.al [36], we take all $\binom{K}{2}$ pairwise comparisons from identical initial conditions (s_t, I) and group these together as a single batch for training.

4.3 Reward Modeling

The loss function for training the reward model follows the Bradley-Terry model [37] with additional modifications to account for preference levels. Formally, we define the ground truth preference level as $\Delta_t^* = \left|\text{RMSE}(a_t^W, a_t^*) - \text{RMSE}(a_t^L, a_t^*)\right|$ and the predicted preference level from our action verifier $R_\phi : A \times S \times \mathcal{I} \to \mathbb{R}$, parameterized by $\phi \in \mathbb{R}^{|\phi|}$, as $\hat{\Delta}_t = \left|R_\phi(a_t^W, s_t, I) - R_\phi(a_t^L, s_t, I)\right|$. These components are then integrated into our loss function for training the reward model:

$$\mathcal{L}(\phi; \mathcal{D}_{\text{comp}}) = -\mathbb{E}_{(a_t^W,\, a_t^L,\, a_t^*,\, s_t,\, I) \sim \mathcal{D}_{\text{comp}}}\left[\log \sigma\left(R_\phi(a_t^W, s_t, I) - R_\phi(a_t^L, s_t, I) - \alpha \left\|\Delta_t^* - \hat{\Delta}_t\right\|_2^2\right)\right],$$

where $\sigma : \mathbb{R} \to [0, 1]$ is the sigmoid function and $\alpha \in \mathbb{R}$ is a hyperparameter to control the magnitude of the preference level. We find that including the margin component $\left\|\Delta_t^* - \hat{\Delta}_t\right\|_2^2$ improves the accuracy, particularly when distinguishing between clearly different actions. For more detailed analysis and ablation studies, please refer to Appendices E and F. The action verifier uses LLaVA-7B [38, 39] as the backbone and replaces its final unembedding layer with a reward head. The architecture integrates ViT-Large [40] as the vision encoder and uses a MLP to map the visual features into the same dimensionality as the word embedding space of the language model.

4.4 Action Sampling and Verification

A visualization of the pipeline is shown in Figure 2. Formally, at each timestep t during deployment under instruction I, RoboMonkey first samples $\hat{N}$ candidate actions from a VLA model $\pi_\theta(a \mid s_t, I; \mathcal{T})$ with a positive temperature $\mathcal{T}$, yielding a set of candidate actions $\hat{A} = \{\hat{a}_t^1, \ldots, \hat{a}_t^{\hat{N}}\} \in \mathbb{R}^{m \times \hat{N}}$. Given these samples, we determine the gripper action g_t via majority voting over the discrete gripper component: $g_t = \text{mode}(\{g_t^i\}_{i=1}^{\hat{N}})$. We then fit a Gaussian distribution $\mathcal{N}(\mu_t, \Sigma_t)$ to both the translational components $\{[\Delta\hat{x}_t^i, \Delta\hat{y}_t^i, \Delta\hat{z}_t^i]'\}_{i=1}^{\hat{N}}$ and rotational components $\{[\Delta\hat{u}_t^i, \Delta\hat{v}_t^i, \Delta\hat{w}_t^i]'\}_{i=1}^{\hat{N}}$. RoboMonkey then samples $\hat{K}$ new actions from this proposal distribution and appends the fixed gripper state g_t to each, forming a refined action set $\tilde{A} = \{\tilde{a}_t^1, \ldots, \tilde{a}_t^{\hat{K}}\} \in \mathbb{R}^{m \times \hat{K}}$. Finally, each action $\tilde{a}_t^i$ is scored using our reward model $R_\phi(\tilde{a}_t^i, s_t, I)$ from which we select the action with the highest reward for execution $a_t = \text{argmax}_{\tilde{a}_t^i \in \{\tilde{a}_t^1, \ldots, \tilde{a}_t^{\hat{K}}\}} R_\phi(\tilde{a}_t^i, s_t, I)$. Below, Algorithm 1 presents our detailed test-time scaling pipeline.

```
Algorithm 1: RoboMonkey Execution
Input: Generic VLA model $\pi_\theta : \mathcal{S} \times \mathcal{I} \to A$, reward model $R_\phi : A \times \mathcal{S} \times \mathcal{I} \to \mathbb{R}$, initial state $s_0 \in \mathcal{S}$,
       task instruction $I \in \mathcal{I}$, temperature $\mathcal{T} \in \mathbb{R}_{++}$, time horizon $T \in \mathbb{N}_+$, number of VLA samples
       $\hat{N} \in \mathbb{N}_+$, number of Gaussian samples $\hat{K} \in \mathbb{N}_+$.
for $t = 0, 1, \ldots, T$ do
    Sample $\hat{A}_t = \{\hat{a}_t^i\}_{i=1}^{\hat{N}} \sim \pi_\theta(a \mid s_t, I; \mathcal{T})$
    Compute gripper state $g_t \leftarrow \text{mode}(\{\hat{g}_t^i\}_{i=1}^{\hat{N}})$
    Fit Gaussian distribution $\mathcal{N}(\mu_t, \Sigma_t)$ on $\{[\Delta\hat{x}_t^i, \Delta\hat{y}_t^i, \Delta\hat{z}_t^i, \Delta\hat{u}_t^i, \Delta\hat{v}_t^i, \Delta\hat{w}_t^i]'\}_{i=1}^{\hat{N}}$
    Sample $\tilde{A}_t = \{\tilde{a}_t^i\}_{i=1}^{\hat{K}} \sim \mathcal{N}(\mu_t, \Sigma_t)$
    Set $\tilde{a}_t^i \leftarrow [\Delta\tilde{x}_t^i, \Delta\tilde{y}_t^i, \Delta\tilde{z}_t^i, \Delta\tilde{u}_t^i, \Delta\tilde{v}_t^i, \Delta\tilde{w}_t^i, g_t]'$ for all $i \in \{1, 2, \ldots, \hat{K}\}$
    Select action $a_t \leftarrow \text{argmax}_{\tilde{a}_t^i \in \{\tilde{a}_t^1, \ldots, \tilde{a}_t^{\hat{K}}\}} R_\phi(\tilde{a}_t^i, s_t, I)$
    Execute $a_t$ and observe $s_{t+1}$
```

5 Experiments

The goal of our experiments is to evaluate the effectiveness of scaling test-time compute for generalist robot policies. We evaluate RoboMonkey across both simulated and real-world environments using two different embodiments, across 4 real-robot tasks and 14 simulation tasks.

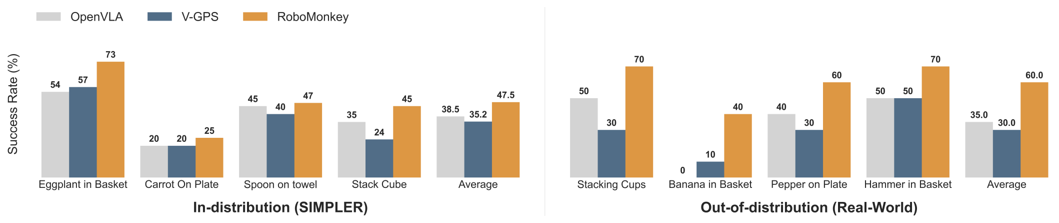

Figure 3: Scaling test-time compute significantly improves the precision and robustness of generalist robot policies across a wide range of manipulation tasks. We observe an 9% increase in average success rate on in-distribution SIMPLER environments [41], and a 25% improvement in real-world out-of-distribution experiments using the WidowX robot.

5.1 Implementation Details

We use the Bridge V2 Dataset [11] as our primary training dataset, which includes over 40,000 real-world robotic trajectories collected using the 6-DoF WidowX robot in 24 distinct environments. Following the procedure described in Section 4.2, we curated a synthetic action preference dataset consisting of 20 million comparisons. Training was conducted on 8 NVIDIA H100 GPUs with a batch size of 256 using LoRA (r=512, α=128). We use OpenVLA as the base model for all experiments. Both RoboMonkey and V-GPS [42] are evaluated by pairing OpenVLA with their respective verifier checkpoints. In real-world evaluation, the system runs at approximately 1.5 Hz on a single NVIDIA H100 GPU and uses a total of 28 GB of GPU memory. For more details about model training and deployment, please refer to Appendix B.

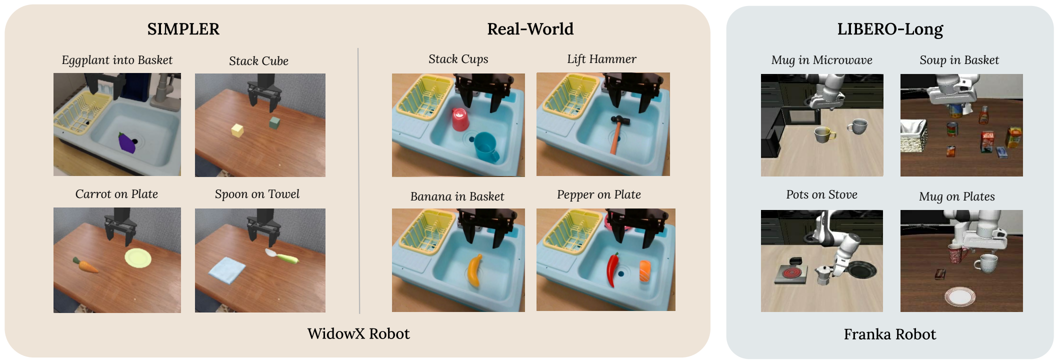

Figure 4: Example tasks across SIMPLER, real-world, and LIBERO environments.

5.2 Can RoboMonkey improve the precision of VLAs on in-distribution tasks?

We first evaluate our model within the SIMPLER [41] environment on in-distribution tasks. This simulation environment is specifically designed to bridge the real-to-sim gap by replicating real-world conditions for WidowX robots. It has demonstrated a strong correlation between performance in SIMPLER and real-world results [41]. We evaluate RoboMonkey and other baselines on four tasks: put eggplant in yellow basket, put carrot on plate, put spoon on towel, and stack green block on yellow block.

Figure 3 presents the evaluation results. RoboMonkey achieves an average success rate of 47.5%, outperforming OpenVLA by 9% on average. In the task of placing an eggplant into a basket, RoboMonkey outperforms OpenVLA by 19%. We observe that the base policy frequently collides with the wall when attempting to move the eggplant toward the basket. Similarly, in the block-stacking task, RoboMonkey surpasses OpenVLA by 10%. This task requires accurate grasping and precise placement of small objects. Overall, these results highlight that making local refinements to the base policy's actions can substantially improve precision. Additionally, we observe that pairing V-GPS with OpenVLA results in worse performance than both standalone OpenVLA and RoboMonkey. See Appendices A and C for details on the task setup and the ablation study on action selection.

5.3 Can RoboMonkey improve the robustness of VLAs on out-of-distribution tasks?

For a more comprehensive evaluation, we design a set of real-world manipulation tasks on a physical WidowX-250 S robot to evaluate RoboMonkey in OOD settings. As illustrated in Figure 4, these tasks include unseen instructions, objects, distractors, and shapes. We evaluate each approach across 4 task suites with 10 trials each, resulting in a total of 120 rollouts. Figure 3 compares the performance of RoboMonkey, V-GPS, and OpenVLA across a suite of tasks on the WidowX robot. RoboMonkey consistently outperforms both baselines across diverse tasks, including stacking cups, lifting a hammer, placing banana into a yellow basket, and putting a pepper onto a plate. We find that RoboMonkey effectively mitigates issues of imprecise grasping, task progression failures, and collisions at deployment. Detailed task breakdowns and failure analysis are provided on our project page: https://robomonkey-vla.github.io.

Notably, RoboMonkey exhibits substantial improvements on tasks requiring visual and semantic generalization. For example, in the banana-in-basket task, OpenVLA achieved a success rate of 0%, as it lacks the language and visual grounding to differentiate between two yellow objects (banana and yellow basket), thus making no progress in completing the task. Furthermore, RoboMonkey achieves over 20% higher success rates on fine-grained manipulation tasks such as cup stacking and hammer lifting. These tasks require precise reasoning about grasp points, particularly on novel objects and shapes. Overall, RoboMonkey achieved an average success rate of 60%, compared to 35% from OpenVLA and 30% from V-GPS, indicating that our action verifier is significantly less sensitive to distribution shifts than the base policy. As such, these results underscore RoboMonkey's effectiveness in improving the robustness and generalization in OOD scenarios.

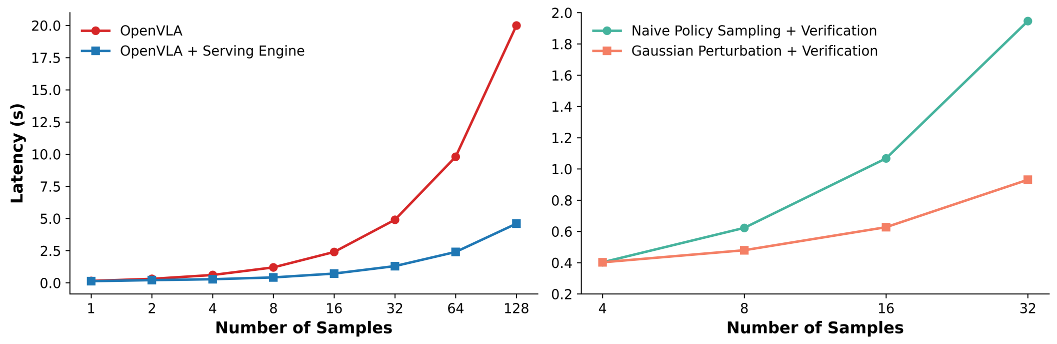

Figure 5: **Left:** Repeated sampling can exploit KV Cache optimizations and batch processing to achieve higher throughput than greedy decoding. Therefore, we extended SGLang's capabilities to properly support OpenVLA. Our optimized implementation substantially outperforms the naive OpenVLA inference pipeline, achieving lower latency and significantly higher throughput across batch sizes. **Right:** Latency comparison between naive policy sampling and Gaussian perturbation as we scale the number of samples.

5.4 How does RoboMonkey enable practical deployment for test-time scaling?

While RoboMonkey introduces additional computational overhead from action sampling and verification, we mitigate these costs with a practical serving solution. Specifically, we implemented a VLA serving engine on top of SGLang [43] to speed up repeated sampling of initial action candidates (see Figure 5) and employ Gaussian perturbation to efficiently construct an action proposal distribution, as detailed in Section 4.4. With these optimizations, RoboMonkey can sample and verify 16 candidate actions in approximately 650 ms (or 1.5 Hz), achieving a 41.3% lower latency compared to naive policy sampling, as shown in Figure 5 (right). Gaussian perturbation proves more efficient because latency scales only with verification cost, whereas naive policy sampling incurs increasing latency from both sampling and verification as the number of candidate actions grows. See Appendix H for a detailed analysis of the trade-off between action error and compute budget.

SIMPLER results were obtained using two NVIDIA RTX 4090 GPUs, while real-world experiments and latency analysis were conducted on a single NVIDIA H100. LIBERO evaluations used an NVIDIA RTX 6000 Ada.

5.5 How does scaling the synthetic training dataset impact downstream success rate?

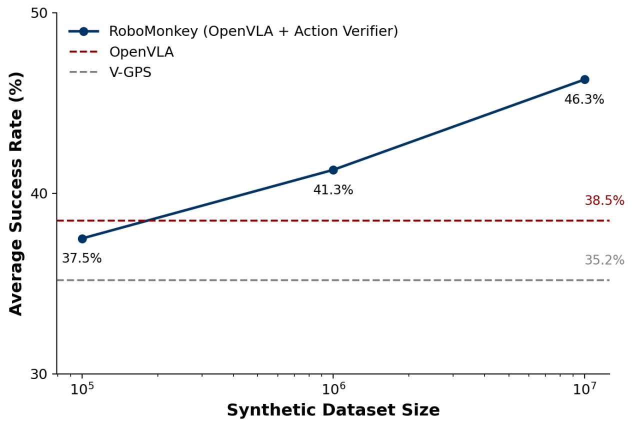

Figure 6: Average success rates across four tasks on SIMPLER as a function of synthetic dataset size. Scaling the dataset size (number of synthetic action comparisons) consistently improves the performance of the RoboMonkey action verifier, leading to higher closed-loop success rates.

We demonstrate that RoboMonkey's closed-loop success rates on SIMPLER consistently improve as the synthetic dataset size increases, with the average success rate rising from 37.5% to 46.3% as shown in Figure 6. With action verifiers trained on over 10^6 action comparisons, RoboMonkey outperforms both OpenVLA and V-GPS. The improvements are particularly pronounced in fine-grained manipulation tasks like "Stacking Cube", where success rates increase from 27% to 37% to 42% as the synthetic dataset scales. We find that the overall task performance grows nearly log-linearly with synthetic dataset size, highlighting the potential of large-scale synthetic data generation for enhancing action verification.

5.6 Can we effectively fine-tune the action verifier on new robot setup and task?

Task	OpenVLA	RoboMonkey
Soup and Sauce in Basket	36%	**59%**
Cheese and Butter in Basket	70%	**79%**
Turn on Stove and Place Moka	58%	**58%**
Black Bowl in Drawer	36%	**37%**
Mugs on Plates	42%	**55%**
Book in Caddy	84%	**86%**
Mug and Pudding on Plate	48%	**59%**
Soup and Cheese in Basket	56%	**62%**
Moka Pots on Stove	26%	**26%**
Mug in Microwave	42%	**44%**
Average Success Rate	49.8%	**56.5%**

Table 1: Comparison of task success rates between OpenVLA and RoboMonkey, both fine-tuned and evaluated on the LIBERO-Long benchmark.

We further evaluate RoboMonkey's adaptability on a new robot setup. In this section, we present the fine-tuning evaluation of RoboMonkey on the LIBERO-Long benchmark, which consists of long-horizon tasks in simulation. In our experiments, we curated a new action preference dataset using the procedure described in Section 4.2 for fine-tuning the action verifier. The OpenVLA-LIBERO checkpoint used for comparison was trained via behavioral cloning on successful demonstrations. All methods were evaluated across 500 trials. Table 1 presents the results and we observe that RoboMonkey can be effectively adapted to tasks in the LIBERO environments. We find that fine-tuning both OpenVLA and action verifier results in 6.7% improvement in average success rate compared to simply fine-tuning OpenVLA on LIBERO-Long.

6 Discussion and Limitations

In this paper, we presented RoboMonkey, a novel test-time scaling framework that enhances the precision and robustness of Vision-Language-Action (VLA) models. RoboMonkey achieves significant performance improvements across both in-distribution and out-of-distribution tasks, as well as on new robot setups. Our findings demonstrate that scaling test-time compute through a generate-and-verify paradigm provides a practical and effective path towards building general-purpose robotics foundation models. The current RoboMonkey framework has several limitations that we leave for future work:

Computational Overhead: Since it requires sampling multiple candidate actions from a VLA and employs a separate VLM-based action verifier, it incurs increased computational overhead during deployment. Although we mitigate these costs with a practical serving solution—using a VLA serving engine and Gaussian perturbation—the framework may still be less suitable for tasks requiring high-frequency control. Future work could explore more efficient model architectures for action verification and apply system-level optimizations to further reduce the memory footprint and latency of scaling test-time compute.

Scaling Synthetic Datasets: Our results show that increasing the size of the synthetic training dataset consistently improves downstream robot performance. However, due to compute constraints and the high cost of fine-tuning, we limited our experiments to 20 million synthetic action comparisons on the Bridge V2 dataset. Scaling synthetic data generation to larger robotics datasets across embodiments, tasks, and environments is a promising direction for future exploration.

Evaluation Scope: While our experiments focused on two commonly used robotic arms—WidowX 250S and Franka—future work should evaluate RoboMonkey across a broader range of embodiments.

7 Related Work

Vision Language Action Models: Recent advancements in robotics have seen a shift toward training multi-task generalist robot policies [44] on large robotics datasets [11, 45] collected on diverse scenes and robot embodiments. In this landscape, several robotics foundation models have emerged. π_0, OpenVLA, PaLM-E, and RT-X [1, 2, 3, 4, 5] have demonstrated strong generalization capabilities by combining Transformer architectures [46] or diffusion policies [13] with imitation learning. While these generalist policies demonstrate out-of-the-box capabilities for controlling robots, they may still fail due to distribution shift and compounding prediction errors. Our evaluation demonstrates that RoboMonkey significantly improves the robustness and generalization of these generalist policies at deployment.

Out-of-Distribution Robustness: The challenge of learning-based systems performing unreliably on data that differs from their training distribution is documented across robotics literature [47, 8, 9]. Researchers have approached this challenge through various methodologies, including robust training and adapting models to varying environmental distribution shifts [47, 48, 49]. A significant breakthrough came with the emergence of Foundation Models (FMs). Recently, FM is widely adopted in robotics. For instance, several prior works [50, 51, 52] explore employing VLMs to generate sequences of high-level action plans, which are then executed through low-level policy. In contrast, rather than using them primarily for hierarchical planning, RoboMonkey and several concurrent works [53, 54] employ FMs as action verifiers that evaluate the low-level actions generated by robot policies.

Repeated Sampling: The methodology of applying additional computation at test time has demonstrated remarkable success across various domains. For LLMs, repeated sampling has proven effective in enhancing performance across diverse tasks, including mathematical problem-solving, coding, and text summarization [28, 22, 55]. In robotics, V-GPS [42] adopts a related strategy by training a value function with offline RL to re-rank candidate actions, selecting those that lead to better outcomes. RoboMonkey introduces a more scalable data curation pipeline and model architecture for training the action verifier. Our experimental results show that pairing existing VLAs with our verifier substantially improves both task performance and generalization compared to prior verifier-based approaches. Instead of relying on naive action sampling from robot policies, RoboMonkey uses Gaussian perturbation to efficiently generate diverse candidate actions and integrates inference-time techniques such as majority voting to guide the verification process.

Acknowledgments

This work was supported by DARPA and the National Aeronautics and Space Administration under the University Leadership Initiative program.

References

[1] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, S. Jakubczak, T. Jones, L. Ke, S. Levine, A. Li-Bell, M. Mothukuri, S. Nair, K. Pertsch, L. X. Shi, J. Tanner, Q. Vuong, A. Walling, H. Wang, and U. Zhilinsky. π_0: A vision-language-action flow model for general robot control, 2024. URL https://arxiv.org/abs/2410.24164.

[2] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

[3] D. Driess, F. Xia, M. S. M. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, W. Huang, Y. Chebotar, P. Sermanet, D. Duckworth, S. Levine, V. Vanhoucke, K. Hausman, M. Toussaint, K. Greff, A. Zeng, I. Mordatch, and P. Florence. Palm-e: An embodied multimodal language model, 2023. URL https://arxiv.org/abs/2303.03378.

[4] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[5] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, X. Chen, K. Choromanski, T. Ding, D. Driess, A. Dubey, C. Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

[6] A. Khazatsky, K. Pertsch, S. Nair, A. Balakrishna, S. Dasari, S. Karamcheti, S. Nasiriany, M. K. Srirama, L. Y. Chen, K. Ellis, et al. Droid: A large-scale in-the-wild robot manipulation dataset. *arXiv preprint arXiv:2403.12945*, 2024.

[7] A. O'Neill, A. Rehman, A. Gupta, A. Maddukuri, A. Gupta, A. Padalkar, A. Lee, A. Pooley, A. Gupta, A. Mandlekar, et al. Open x-embodiment: Robotic learning datasets and rt-x models. *arXiv preprint arXiv:2310.08864*, 2023.

[8] C. Agia, R. Sinha, J. Yang, Z. ang Cao, R. Antonova, M. Pavone, and J. Bohg. Unpacking failure modes of generative policies: Runtime monitoring of consistency and progress, 2024. URL https://arxiv.org/abs/2410.04640.

[9] R. Sinha, A. Elhafsi, C. Agia, M. Foutter, E. Schmerling, and M. Pavone. Real-time anomaly detection and reactive planning with large language models, 2024. URL https://arxiv.org/abs/2407.08735.

[10] Z. Zhou, P. Atreya, A. Lee, H. Walke, O. Mees, and S. Levine. Autonomous improvement of instruction following skills via foundation models, 2024. URL https://arxiv.org/abs/2407.20635.

[11] H. R. Walke, K. Black, T. Z. Zhao, Q. Vuong, C. Zheng, P. Hansen-Estruch, A. W. He, V. Myers, M. J. Kim, M. Du, et al. Bridgedata v2: A dataset for robot learning at scale. In *Conference on Robot Learning*, pages 1723–1736. PMLR, 2023.

[12] J. Hejna, C. Bhateja, Y. Jiang, K. Pertsch, and D. Sadigh. Re-mix: Optimizing data mixtures for large scale imitation learning, 2024. URL https://arxiv.org/abs/2408.14037.

[13] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, page 02783649241273668, 2023.

[14] C.-L. Cheang, G. Chen, Y. Jing, T. Kong, H. Li, Y. Li, Y. Liu, H. Wu, J. Xu, Y. Yang, H. Zhang, and M. Zhu. Gr-2: A generative video-language-action model with web-scale knowledge for robot manipulation. *arXiv preprint arXiv:2410.06158*, 2024.

[15] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn. Learning fine-grained bimanual manipulation with low-cost hardware, 2023. URL `https://arxiv.org/abs/2304.13705`.

[16] M. Zawalski, W. Chen, K. Pertsch, O. Mees, C. Finn, and S. Levine. Robotic control via embodied chain-of-thought reasoning, 2025. URL `https://arxiv.org/abs/2407.08693`.

[17] J. Clark, S. Mirchandani, D. Sadigh, and S. Belkhale. Action-free reasoning for policy generalization, 2025. URL `https://arxiv.org/abs/2502.03729`.

[18] Q. Zhao, Y. Lu, M. J. Kim, Z. Fu, Z. Zhang, Y. Wu, Z. Li, Q. Ma, S. Han, C. Finn, A. Handa, M.-Y. Liu, D. Xiang, G. Wetzstein, and T.-Y. Lin. Cot-vla: Visual chain-of-thought reasoning for vision-language-action models, 2025. URL `https://arxiv.org/abs/2503.22020`.

[19] Z. Zhang, K. Zheng, Z. Chen, J. Jang, Y. Li, S. Han, C. Wang, M. Ding, D. Fox, and H. Yao. Grape: Generalizing robot policy via preference alignment, 2025. URL `https://arxiv.org/abs/2411.19309`.

[20] B. Zhang, Y. Zhang, J. Ji, Y. Lei, J. Dai, Y. Chen, and Y. Yang. Safevla: Towards safety alignment of vision-language-action model via safe reinforcement learning, 2025. URL `https://arxiv.org/abs/2503.03480`.

[21] D. Li, J. Ren, Y. Wang, X. Wen, P. Li, L. Xu, K. Zhan, Z. Xia, P. Jia, X. Lang, N. Xu, and H. Zhao. Finetuning generative trajectory model with reinforcement learning from human feedback, 2025. URL `https://arxiv.org/abs/2503.10434`.

[22] B. Brown, J. Juravsky, R. Ehrlich, R. Clark, Q. V. Le, C. Ré, and A. Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

[23] C. Snell, J. Lee, K. Xu, and A. Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

[24] J. Saad-Falcon, A. G. Lafuente, S. Natarajan, N. Maru, H. Todorov, E. Guha, E. K. Buchanan, M. Chen, N. Guha, C. Ré, et al. Archon: An architecture search framework for inference-time techniques. *arXiv preprint arXiv:2409.15254*, 2024.

[25] L. Chen, J. Q. Davis, B. Hanin, P. Bailis, I. Stoica, M. Zaharia, and J. Zou. Are more llm calls all you need? towards scaling laws of compound inference systems. *arXiv preprint arXiv:2403.02419*, 2024.

[26] Y. Song, G. Wang, S. Li, and B. Y. Lin. The good, the bad, and the greedy: Evaluation of llms should not ignore non-determinism. *arXiv preprint arXiv:2407.10457*, 2024.

[27] D. Li, S. Cao, C. Cao, X. Li, S. Tan, K. Keutzer, J. Xing, J. E. Gonzalez, and I. Stoica. S*: Test time scaling for code generation, 2025. URL `https://arxiv.org/abs/2502.14382`.

[28] G. Chen, M. Liao, C. Li, and K. Fan. Alphamath almost zero: process supervision without process. *arXiv preprint arXiv:2405.03553*, 2024.

[29] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[30] R. Schaeffer, J. Kazdan, J. Hughes, J. Juravsky, S. Price, A. Lynch, E. Jones, R. Kirk, A. Mirhoseini, and S. Koyejo. How do large language monkeys get their power (laws)?, 2025. URL `https://arxiv.org/abs/2502.17578`.

[31] S. Sartor and N. Thompson. Neural scaling laws in robotics, 2025. URL https://arxiv.org/abs/2405.14005.

[32] F. Lin, Y. Hu, P. Sheng, C. Wen, J. You, and Y. Gao. Data scaling laws in imitation learning for robotic manipulation, 2025. URL https://arxiv.org/abs/2410.18647.

[33] Q. Li, Y. Liang, Z. Wang, L. Luo, X. Chen, M. Liao, F. Wei, Y. Deng, S. Xu, Y. Zhang, X. Wang, B. Liu, J. Fu, J. Bao, D. Chen, Y. Shi, J. Yang, and B. Guo. Cogact: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation, 2024. URL https://arxiv.org/abs/2411.19650.

[34] O. M. Team, D. Ghosh, H. Walke, K. Pertsch, K. Black, O. Mees, S. Dasari, J. Hejna, T. Kreiman, C. Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

[35] D. Qu, H. Song, Q. Chen, Y. Yao, X. Ye, Y. Ding, Z. Wang, J. Gu, B. Zhao, D. Wang, and X. Li. Spatialvla: Exploring spatial representations for visual-language-action model, 2025. URL https://arxiv.org/abs/2501.15830.

[36] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe. Training language models to follow instructions with human feedback, 2022. URL https://arxiv.org/abs/2203.02155.

[37] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[38] H. Liu, C. Li, Q. Wu, and Y. J. Lee. Visual instruction tuning, 2023.

[39] Z. Sun, S. Shen, S. Cao, H. Liu, C. Li, Y. Shen, C. Gan, L.-Y. Gui, Y.-X. Wang, Y. Yang, K. Keutzer, and T. Darrell. Aligning large multimodal models with factually augmented rlhf, 2023. URL https://arxiv.org/abs/2309.14525.

[40] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision, 2021. URL https://arxiv.org/abs/2103.00020.

[41] X. Li, K. Hsu, J. Gu, K. Pertsch, O. Mees, H. R. Walke, C. Fu, I. Lunawat, I. Sieh, S. Kirmani, S. Levine, J. Wu, C. Finn, H. Su, Q. Vuong, and T. Xiao. Evaluating real-world robot manipulation policies in simulation. *arXiv preprint arXiv:2405.05941*, 2024.

[42] M. Nakamoto, O. Mees, A. Kumar, and S. Levine. Steering your generalists: Improving robotic foundation models via value guidance, 2025. URL https://arxiv.org/abs/2410.13816.

[43] L. Zheng, L. Yin, Z. Xie, C. Sun, J. Huang, C. H. Yu, S. Cao, C. Kozyrakis, I. Stoica, J. E. Gonzalez, C. Barrett, and Y. Sheng. Sglang: Efficient execution of structured language model programs, 2024. URL https://arxiv.org/abs/2312.07104.

[44] Y. Jiang, A. Gupta, Z. Zhang, G. Wang, Y. Dou, Y. Chen, L. Fei-Fei, A. Anandkumar, Y. Zhu, and L. Fan. Vima: General robot manipulation with multimodal prompts. *arXiv preprint arXiv:2210.03094*, 2(3):6, 2022.

[45] H.-S. Fang, H. Fang, Z. Tang, J. Liu, J. Wang, H. Zhu, and C. Lu. Rh20t: A robotic dataset for learning diverse skills in one-shot. In *RSS 2023 Workshop on Learning for Task and Motion Planning*, 2023.

[46] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. advances in neural information processing systems. *Advances in neural information processing systems*, 30(2017), 2017.

[47] R. Sinha, A. Sharma, S. Banerjee, T. Lew, R. Luo, S. M. Richards, Y. Sun, E. Schmerling, and M. Pavone. A system-level view on out-of-distribution data in robotics, 2023. URL `https://arxiv.org/abs/2212.14020`.

[48] E. Zhu, M. Levy, M. Gwilliam, and A. Shrivastava. Nerf-aug: Data augmentation for robotics with neural radiance fields, 2025. URL `https://arxiv.org/abs/2411.02482`.

[49] P. Mitrano and D. Berenson. Data augmentation for manipulation, 2022. URL `https://arxiv.org/abs/2205.02886`.

[50] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as policies: Language model programs for embodied control, 2023. URL `https://arxiv.org/abs/2209.07753`.

[51] Y. Mu, Q. Zhang, M. Hu, W. Wang, M. Ding, J. Jin, B. Wang, J. Dai, Y. Qiao, and P. Luo. Embodiedgpt: Vision-language pre-training via embodied chain of thought, 2023. URL `https://arxiv.org/abs/2305.15021`.

[52] L. X. Shi, B. Ichter, M. Equi, L. Ke, K. Pertsch, Q. Vuong, J. Tanner, A. Walling, H. Wang, N. Fusai, A. Li-Bell, D. Driess, L. Groom, S. Levine, and C. Finn. Hi robot: Open-ended instruction following with hierarchical vision-language-action models, 2025. URL `https://arxiv.org/abs/2502.19417`.

[53] Y. Wu, R. Tian, G. Swamy, and A. Bajcsy. From foresight to forethought: Vlm-in-the-loop policy steering via latent alignment, 2025. URL `https://arxiv.org/abs/2502.01828`.

[54] Y. Wang, L. Wang, Y. Du, B. Sundaralingam, X. Yang, Y.-W. Chao, C. Perez-D'Arpino, D. Fox, and J. Shah. Inference-time policy steering through human interactions, 2025. URL `https://arxiv.org/abs/2411.16627`.

[55] R. Ehrlich, B. Brown, J. Juravsky, R. Clark, C. Ré, and A. Mirhoseini. Codemonkeys: Scaling test-time compute for software engineering, 2025. URL `https://arxiv.org/abs/2501.14723`.

[56] B. Liu, Y. Zhu, C. Gao, Y. Feng, Q. Liu, Y. Zhu, and P. Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *arXiv preprint arXiv:2306.03310*, 2023.

A Evaluation Tasks

As described in Section 5.3 and illustrated in Figure 4, we evaluate RoboMonkey, OpenVLA, and V-GPS on a physical WidowX-250 S robot across four out-of-distribution (OOD) generalization tasks. Real-world evaluations inherently introduce distribution shifts, as we cannot exactly replicate the Bridge V2 setup. In particular, slight variations in camera placement, robot positioning, lighting conditions, and background are unavoidable. We describe the four representative generalization tasks as follows:

- **Stack Blue Cup on Pink Cup**: The goal is to grasp the blue cup and stack it on top of the pink cup. The language instruction of this task was not included in the Bridge V2 dataset.
- **Put Hammer into Yellow Basket**: The robot must lift the hammer and place it inside a yellow basket. Importantly, the hammer represents an unseen object with a novel shape not present in any Bridge V2 demonstration.
- **Put Pepper onto Plate**: This language grounding task requires the robot to identify and approach the pepper while ignoring distractors (e.g., sushi). The robot must then differentiate between the yellow basket and a plate before correctly placing the pepper.
- **Put Banana into Yellow Basket**: The objective of this task is to place a banana from the sink into the yellow basket. This task presents a particular challenge as the system must differentiate between two yellow objects (banana and yellow basket), requiring visual and language grounding to complete the task successfully.

For details on the task setup for in-distribution and fine-tuning evaluation, please refer to the SIMPLER [41] and LIBERO-LONG [56] benchmark. We include task execution examples in Figure 7.

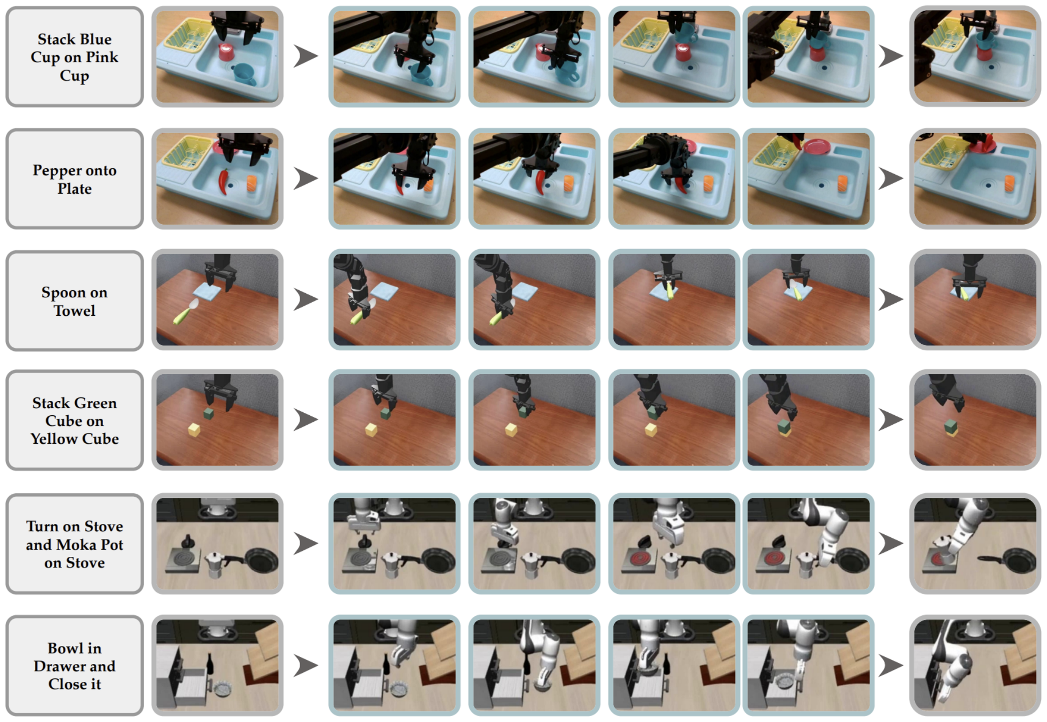
Stack Blue Cup on Pink Cup
Pepper onto Plate
Spoon on Towel
Stack Green Cube on Yellow Cube
Turn on Stove and Moka Pot on Stove
Bowl in Drawer and Close it

Figure 7: Representative task executions in real-world, SIMPLER, and LIBERO environments.

B Implementation Details

B.1 Training

Our action verifier uses LLaVA-7B [38, 39] as the backbone and replaces its final unembedding layer with a reward head. The architecture integrates ViT-Large [40] as the vision encoder and uses a MLP to map the visual features into the same dimensionality as the word embedding space of the language model. We discretize each dimension of a continuous action into 256 bins, following Brohan et al. [4], and overwrite the 256 least frequent tokens in the LLaMA tokenizer with these discrete action tokens.

Training was conducted on 8 NVIDIA H100 GPUs using LoRA (rank=512, $\alpha = 128$), and our codebase builds on top of LLaVA-RLHF [39]. We use a batch size of 256 and train on a synthetic preference dataset comprising 20 million comparisons derived from the Bridge V2 dataset. The model is trained using the Adam optimizer with a learning rate of 2e-5. Training is conducted for a single epoch. A margin weight of 0.1 is applied to the modified Bradley-Terry loss. Example prompt to action verifier is shown below:

```
USER: <image> shows the current observation from the robot's wrist-
mounted camera. The robot manipulation arm is attempting to [instruction].
What action should the robot take to effectively accomplish the task?
ASSISTANT: The robot should take the action [Discrete Action Tokens]
USER: Please evaluate the quality of the robot action.
ASSISTANT: The quality score of the robot action is
```

B.2 Deployment

For real-world evaluation, we first sample 5 initial actions from OpenVLA with temperature 1.0. We fit a Gaussian distribution $\mathcal{N}(\mu,\sigma)$ to the translation and rotation components, and use majority voting to determine the gripper state. This creates an action proposal distribution from which we sample 16 candidate actions. We then use the fine-tuned VLM-based verifier to select the optimal action for execution. We conduct 10 trials per task and report the average success rate. In simulation, we vary the number of initial action samples $\hat{N} \in \{5,9\}$ and the number of augmented samples $\hat{K} \in \{8,16,32\}$. We report the best results for each task. All simulated experiments are conducted using a machine equipped with two NVIDIA RTX 4090 GPUs over three random seeds.

B.3 Baselines

We use the publicly released OpenVLA checkpoint from `https://huggingface.co/openvla/openvla-7b`, and the V-GPS value function checkpoint from `https://github.com/nakamotoo/V-GPS`. In simulation, we follow the evaluation procedure outlined in the V-GPS implementation, sweeping over the number of samples $\{10,50\}$ and softmax temperatures $\{0,0.1,1.0\}$, and report the best result for each task. For real-world evaluations, we fix the number of samples to 10 and the temperature to 1.0. It is worth noting that we use our VLA serving engine to enable efficient batch inference for all experiments.

C Ablation Over Action Selection Methods and Number of Samples

To evaluate the effectiveness of our verifier, we adopt a setup similar to that described in Section 3. Specifically, we uniformly sample 1,000 (s,a^*,I) tuples from the auxiliary dataset $\mathcal{D}_{\text{buf}}$, curated from the BridgeV2 [11]. For each tuple, we generate 64 candidate actions using the reference policy, OpenVLA, and apply various selection techniques—including RoboMonkey, V-GPS, majority voting, and random selection—to identify the optimal action among the samples. We report the normalized RMSE between the ground-truth action and the selected action for each method. As shown in Figure 8, RoboMonkey consistently achieves the lowest action error across different sample sizes. When generating 64 samples, RoboMonkey reduces the action error by 21% relative to the greedy decoding baseline, highlighting the effectiveness of our verifier in improving action precision. While prior work such as V-GPS trained with offline RL also improves over greedy decoding, its value function achieves only a 6% reduction in action error. Furthermore, we observe that increasing the number of samples leads to exploitation of the V-GPS value

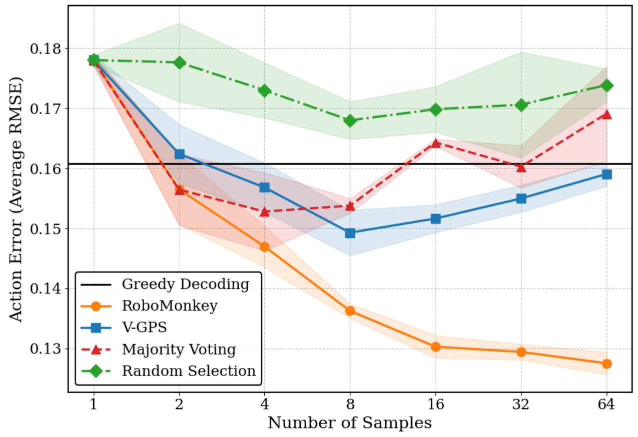

Figure 8: Comparison of action error (average RMSE) across different selection methods as the number of generated samples increases. RoboMonkey consistently outperforms other baselines and scales effectively with additional compute.

function, resulting in performance degradation when sampling more than 8 actions. In contrast, RoboMonkey remains robust to reward hacking and demonstrates scalability with increased test-time compute.

D Ablation Over Generalist Robot Policies

We conducted additional experiments to ablate the performance of RoboMonkey when paired with different VLA models. For models that generate action chunks, we apply temporal ensembling and discretize the outputs using the scheme introduced by Brohan et al. [5] to enable scoring by the action verifier. Following a similar evaluation setup to Appendix C, our ablation considers three generalist robot policies: CogACT, Octo, and SpatialVLA.

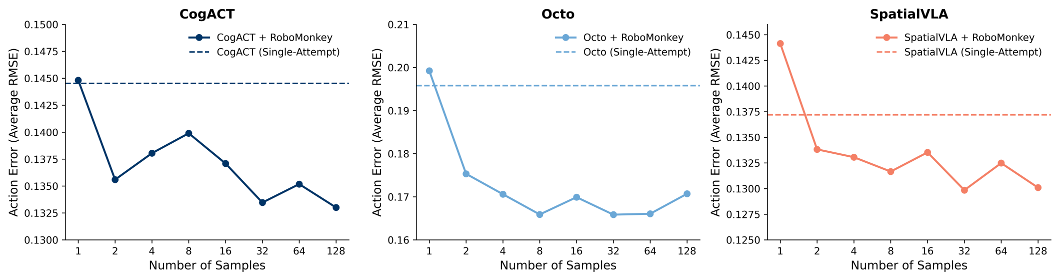

Figure 9: Effect of scaling test-time compute with RoboMonkey across different generalist robot policies. Action error (average RMSE) decreases as the number of samples increases. Dashed lines denote the action error of each base policy when only generating a single action.

CogACT is a 7B-parameter VLA model with a modular architecture that separates cognitive reasoning from motor control. Built on top of the Prismatic VLM (DINOv2 + SigLIP for vision and LLaMA-2 for language), CogACT introduces a specialized action module implemented as a diffusion transformer. We use the CogACT-base variant in our experiments. Pairing RoboMonkey with CogACT achieves RMSE of 0.133, reflecting an 8% reduction from its single-attempt baseline of 0.145.

Octo is a transformer-based generalist policy trained on 800K demonstrations from the Open X-Embodiment (OXE) dataset. The policy includes a CNN encoder and a ViT-style transformer backbone with a diffusion-based action head that predicts action sequences. We use Octo-small for evaluation. Integrating RoboMonkey with Octo achieves RMSE of 0.166, representing a 15.3% reduction from its greedy baseline (0.196).

SpatialVLA is a 3.5B-parameter spatially grounded VLA model trained on 1.1M robot episodes from OXE and RH20T. The model uses Ego3D Position Encoding to integrate 3D spatial context from depth estimates into visual features. It is pre-trained on a PaLI-Gemma-2 backbone. For deployment, we use a temperature of 0.5 for sampling. SpatialVLA + RoboMonkey achieves RMSE of 0.1298, a 5.3% reduction from its baseline of 0.137.

E Ablation on Margin for Reward Modeling

α	Precision	Recall	F1 Score
0	0.81	0.85	0.83
0.1	**0.84**	**0.87**	**0.85**
1.0	0.79	0.83	0.81

Table 2: Comparison of action verifier performance across different margin weights α in the loss function. We find that incorporating a small margin ($\alpha = 0.1$) improves precision, recall, and F1 score

As illustrated in Section 4.3, the loss function for training the action verifier follows the Bradley-Terry model [37] with an additional margin component to account for preference levels.

$$\mathcal{L}(\phi;\mathcal{D}_{\text{comp}}) = -\mathbb{E}_{(a_t^W,\, a_t^L,\, a_t^*,\, s_t,\, I)\sim\mathcal{D}_{\text{comp}}}\left[\log\sigma\left(R_\phi(a_t^W,\, s_t,\, I) - R_\phi(a_t^L,\, s_t,\, I) - \alpha\left\|\Delta_t^* - \hat{\Delta}_t\right\|_2^2\right)\right],$$

where $\alpha \in \mathbb{R}$ is a hyperparameter to control the magnitude of the preference level. To evaluate the effectiveness of this margin component, we generated 10,000 synthetic action comparison pairs from the Bridge V2 dataset following the procedure outlined in Section 4.2. Each action pair consists of distinctly different actions. We trained two variants of the action verifier with margin terms ($\alpha \in 0.1, 1.0$) and compared their performance to a baseline model without a margin term ($\alpha = 0$). Table 2 reports the precision, recall, and F1 score for each setting. The variant with $\alpha = 0.1$ achieved the highest F1 score (0.85), outperforming both the baseline without a margin ($\alpha = 0$, F1 = 0.83) and the large-margin variant ($\alpha = 1.0$, F1 = 0.81). These results suggest that incorporating a margin term can improve the action verifier's performance, but an excessively large margin may negatively impact verification accuracy. Based on this analysis, we adopt the $\alpha = 0.1$ variant for deployment.

F Ablation Over Preference-Based Learning

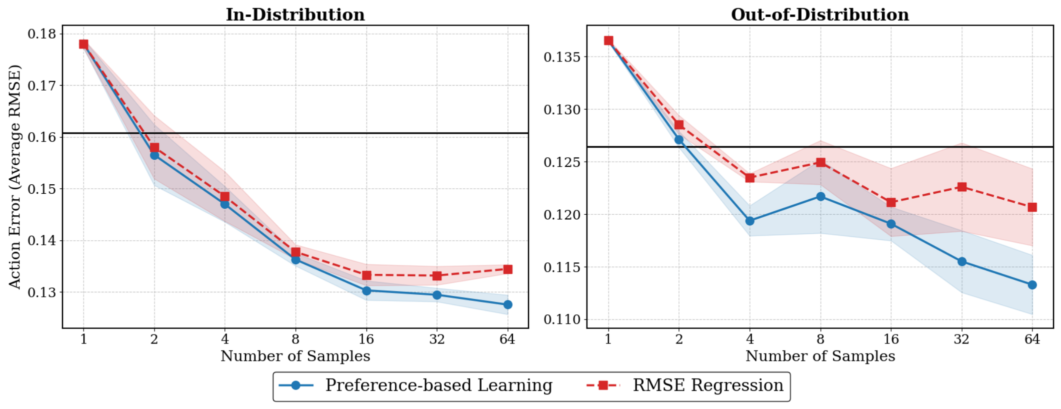

Figure 10: Comparison between preference-based learning and RMSE regression across number of samples. While both perform similarly in-distribution, preference learning generalizes better in OOD settings.

To further understand the effectiveness of our preference-based learning approach, we conduct an ablation comparing our Bradley-Terry objective against a baseline that directly predicts RMSE values. Specifically, we train an alternative action verifier for one epoch to minimize the L2 loss between the predicted and ground-truth RMSE:

$$\mathcal{L}(\phi;\mathcal{D}_{\text{comp}}) = \mathbb{E}_{(a_t, a_t^*, s_t, I)\sim\mathcal{D}_{\text{comp}}}\|R_\phi(a_t, s_t, I) - \text{RMSE}(a_t, a_t^*)\|_2^2$$

where the verifier directly learns to predict the RMSE between any candidate action a_t and the ground-truth action a_t^*.

We follow the same setup as described in Section 3 and Appendix C for in-distribution evaluation using the Bridge V2 dataset [11]. For OOD analysis, we sample 1,000 (s,a^*,I) tuples from held-out trajectories with unseen instructions and environments. The error bars represent the standard deviation across three random seeds.

As shown in Figure 10, both approaches achieve similar performance on in-distribution environments, with preference-based learning slightly outperforming RMSE regression. However, the performance gap becomes substantial in OOD settings. When sampling 64 candidate actions, preference-based learning achieves a 6% lower action error compared to RMSE regression. This result reveals a crucial insight: instead of directly regressing RMSE values, preference-based learning teaches the model to make relative comparisons between actions, enabling stronger generalization.

G Latency and Throughput Analysis for Sampling and Verification

Batch	OpenVLA		OpenVLA + SGLang		Action Verifier	
Sizes	Latency (s)	Throughput	Latency (s)	Throughput	Latency (s)	Throughput
1	0.15	6.5	0.13	7.6	0.092	11
2	0.31	3.3	0.21	9.6	0.099	20
4	0.61	1.6	0.28	15	0.13	32
8	1.2	0.82	0.42	19	0.20	39
16	2.4	0.41	0.72	22	0.35	46
32	4.9	0.20	1.3	25	0.65	49
64	9.8	0.10	2.4	27	1.3	50
128	20	0.050	4.6	28	2.5	51

Table 3: Latency (seconds) and throughput (samples/second) comparison across batch sizes for OpenVLA, Optimized OpenVLA, and 7B Action Verifier.

Repeated sampling can exploit KV Cache optimizations and batch processing to achieve higher throughput than greedy decoding. However, most VLA models, including OpenVLA, are built on top of Prismatic VLM and do not support batching [2]. SGLang provides efficient serving with prefix caching, overhead-free CPU scheduling, and paged attention. Therefore, to make RoboMonkey practical for deployment, we extended SGLang's capabilities to properly support Prismatic VLM [2] models, enabling us to achieve higher throughput during repeated sampling. Users can easily port their Prismatic VLM models to SGLang using our provided template.

We conducted experiments on a single H100 to measure the latency and throughput of OpenVLA inference across varying batch sizes. As shown in Table 3, our optimized implementation significantly outperforms the naive version [2]. For instance, at a batch size of 32, our serving engine reduces latency by 74% and increases throughput by over 120x. Even at smaller batch sizes (e.g. 4), our VLA serving engine achieves a 54% reduction in latency.

Our action verifier also benefits significantly from batch inference, achieving a throughput of 46 actions/s at a batch size of 16. It is notably faster than OpenVLA, as it only requires computation during the prefill stage, which can fully leverage GPU parallelism. In contrast, OpenVLA involves both the prefill and decode stages—where action tokens must be generated autoregressively—resulting in lower throughput.

H Trade-off between Action Error and Computational Overhead

In this ablation study, we examine the trade-off between action error and computational overhead. In Figure 11, we reproduce the scaling curve from Section 3, but plotting action error against latency. Under equivalent computation budgets, Gaussian perturbation achieves significantly lower oracle action error compared to policy sampling. We observe that the gap between the two methods widens as compute budgets increase. This growing performance gap reflects the fact that Gaussian perturbation scales only

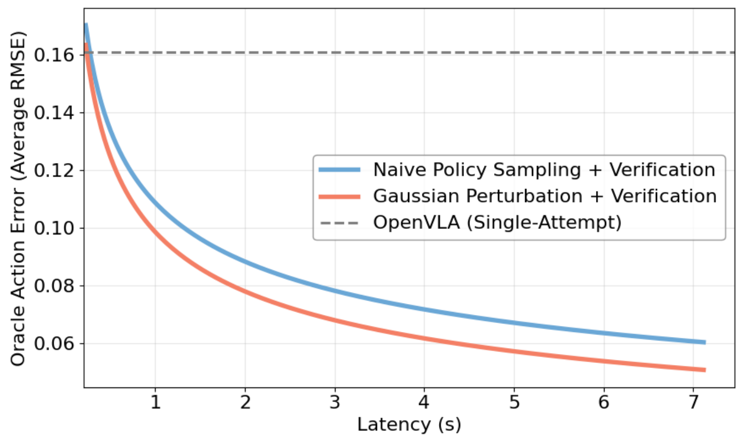

Figure 11: Action error (average RMSE) as a function of computational overhead for policy sampling and Gaussian perturbation. Gaussian perturbation consistently achieves lower action error under equivalent computational budgets

with the verifier, while policy sampling incurs additional overhead from both the policy and verifier. These results highlight Gaussian perturbation as a more practical choice for deployment.

I Notation

Symbol	**Description**
$\mathcal{S}$	State space
$\mathcal{A}$	Action space
τ	Trajectory
I	Task instruction
π_θ	Generalist robot policy parameterized by θ
a_t	Action at timestep t: $[\Delta x_t, \Delta y_t, \Delta z_t, \Delta u_t, \Delta v_t, \Delta w_t, g_t]$
g_t	Binary gripper state (open or close)
$P(s' \mid s, a)$	Transition dynamics
$R_\phi(s, a, I)$	Reward model parameterized by ϕ
$\mathcal{D}$	Demonstration dataset
$\mathcal{D}_{\text{buf}}$	Auxiliary dataset of state-action-instruction tuples
$\mathcal{D}_{\text{comp}}$	Synthetic action preference dataset
σ	Sigmoid function
N, K	Number of candidate and clustered actions (training)
a_t^i	i-th action sampled from policy (training)
$\hat{N}, \hat{K}$	Number of sampled and augmented actions (test-time)
$\hat{a}_t^i$	i-th action sampled from policy (test-time)
$\tilde{a}_t^i$	i-th action sampled from proposal distribution
μ_t, Σ_t	Mean and covariance of a Gaussian distribution
(a_t^W, a_t^L)	Preferred and less preferred action pair
Δ_t^*	Ground-truth RMSE difference
$\hat{\Delta}_t$	Predicted RMSE difference
α	Margin weight in loss function